UCLouvain - EPL

LINFO1341

Réseaux informatiques

Rapport - PROJET 1

Professeur :
Bonaventure Olivier

Assistant :
De Coninck Quentin
Piraux Maxime

Auteurs :
Hennebo Eliot - 43762000
Ben Haddou Mehdi - 19912000

1 Introduction

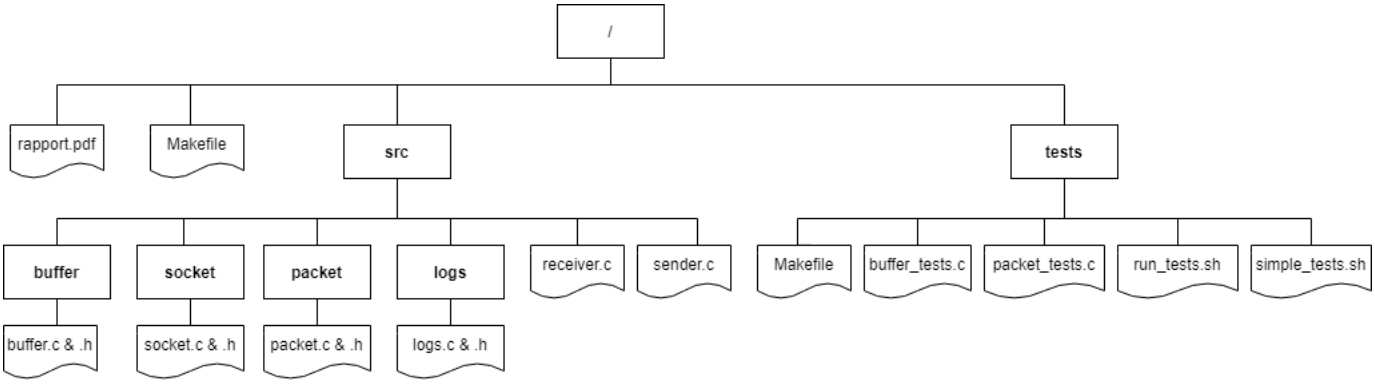
Dans le cadre de ce projet, il nous a été demandé de réaliser une implémentation d'un protocole de transfert de données sur un réseau. Ce protocole TRTP (Truncated Reliable Transfer Protocol), basé sur des paquets UDP fonctionne au dessus du protocole IPv6 et doit permettre de transférer de manière fiable des fichiers.

Ce projet se divise donc principalement en trois parties :

1. **Sender**
2. **Receiver**
3. **Librairies utilitaires**

Nous allons dans ce rapport expliquer la structure du projet, tenter de détailler nos choix de conception, fournir un résumé des différents tests que nous avons pu effectuer dont notamment les tests d'intéropérabilité, et enfin évaluer la robustesse de notre implémentation.

2 Structure des dossiers



Nous avons décidé de structurer ce projet d'une façon assez triviale, notamment en abstrayant un maximum de fonctionnalités dans des librairies utilitaires :

1. **Makefile** : Permet de créer les exécutables dont principalement sender et receiver, la commande tests permet de lancer les tests généraux ainsi que les tests unitaires.
2. **src** : Tout le code source de l'implémentation du protocole.
   - **buffer** : Une librairie permettant de créer et gérer une liste doublement chainée. C'est cette structure de données que nous utilisons principalement dans notre implémentation.
   - **socket** : Une librairie permettant de créer et gérer les sockets, ce code est inspiré d'une tâche INGInious. https://inginious.info.ucl.ac.be/course/LINGI1341/envoyer-et-recevoir-des-donnees
   - **packet** : Une librairie permettant de formatter des paquets UDP, ce code a été réalisé avec l'aide de la tâche INGInious prévue à cet effet. https://inginious.info.ucl.ac.be/course/LINGI1341/format-des-segments
   - **logs** : Fonctions permettant d'afficher des logs, ces fichiers ont été fournis avec le modèle du projet et n'ont pas été particulièrement modifiés.
3. **tests** : Ce dossier contient notre suite de tests unitaires ainsi que les tests généraux fournis par les assistants.

3 Questions de conception

3.1 Buffer : Structure de données

Une des fondations du projet est sans aucun doute le buffer, que ce soit du côté sender ou receiver. Ce buffer est une liste doublement chainée ordonée qui implémente plusieurs fonctions utilitaires relativement classiques mais qui permettent de faciliter grandement plusieurs tâches telles que la suppression de tous les paquets stockés lors de la réception d'un acquittement cumulatif chez le sender par exemple.

Du côté du sender, le buffer contient les paquets qui ont été lus dans le fichier et envoyés, mais qui n'ont pas encore été acquittés. Cette structure de donnée s'avère utile dans ce cas car elle permet de gérer la retransmission des paquets très simplement, nous avons créé une méthode qui vérifie les paquets du buffer ayant dépassé leur timer de retransmission, afin de les renvoyer ; et nous pouvons aussi très facilement renvoyer un paquet si celui-ci est exigé par le receiver. Dans le même temps, c'est très facile de supprimer du buffer l'ensemble des paquets qui ont été acquittés, cela améliore les performances et la gestion de la mémoire.

Du côté du receiver, le buffer contient tous les paquets ayant été bien reçus et étant dans la fenêtre de réception mais qui n'ont pas encore été écrits dans le fichier. Cela facilite grandement le selective repeat, lorsque le receiver doit attendre un paquet avec un numéro de séquence particulier qui est manquant, il peut conserver tous les paquets suivants, écrire dans le fichier le contenu de tous les paquets d'un seul coup lorsque le paquet manquant arrive, et supprimer les paquets traités du buffer.

3.2 Fenêtre de réception

3.3 Génération des acquittements

Nous tirons parti de l'avantage du cumulative acknowledgment pour essayer de réduire la congestion du traffic. Pour ce faire nous envoyons donc un paquet ACK lorsque un de ces cas se présente ;

1. Au moins WINDOW / 4 paquets ont été reçus sans ACK envoyé entre temps.
2. 500ms ce sont écoulé depuis le dernier ACK
3. Un "problème" est survenu tels que ; paquet dupliqué, pas dans la window, tronqué ou encore corrompue.

Le numéro de séquence envoyé dans le paquet ACK correspond au prochain paquet attendu.Dans le cas où le receiver attend un paquet spécifique pour pouvoir écrire les paquets en attente dans le buffer, lorsqu'il reçoit le paquet attendu, le receiver écrit tous les paquets en séquences et envoie un ACK correspondant au prochain paquet attendu.

3.4 Fermeture de la connexion

Le sender signale la fin de son fichier au receiver en envoyant un paquet de type DATA avec une length à 0 et avec le numéro de séquence du dernier paquet DATA envoyé. Le receiver peut donc savoir quand est-ce que le sender à fini d'envoyer des paquets même si la transmission n'est pas fini des deux côtés. Le sender ne clos la liaison que lorsqu'il reçoit l'ACK du dernier seqnum envoyé et le receiver uniquement lorsque il a envoyé l'ACK correspondant au paquet de fin de transmission reçu.

Dans le cas où le dernier paquet de fin de transmission serait perdu, le sender autorise le dernier paquet de timeout 4 fois avant de clôre la transmission. Le sender et le receiver clorent également la connection lorsqu'ils n'ont rien reçu depuis plus de 10 secondes.

3.5 Utilisation du champ TIMESTAMP

Le champ TIMESTAMP sert principalement au sender, lorsque celui-ci envoie un paquet, il remplit le champ avec l'heure Posix (en secondes) et ajoute ce paquet dans son buffer. Lorsque le socket est disponible en écriture, le sender vérifie dans le buffer si un paquet n'a pas dépassé son timer de retransmission. Dans le cas ou un paquet a timed out, celui ci est retransmis en priorité et nous changeons le timestamp de tous les autres paquets qui allaient timed out au même moment par NOW - RTO / 2.

Nous faisons cela pour éviter de renvoyer instantanément tous les paquets en attente d'ACK car dans la majorité des cas, lorsqu'un paquet timeout, le receiver n'attend en réalité que le premier paquet qui timeout. Pour éviter de perdre trop de temps lorsque plusieurs paquets ont été perdus nous ne remettons pas le timer à 0 mais nous le divisons par 2. Cette technique a certaines limites d'efficacité lorsque de nombreux paquets à la suite sont perdus mais nous estimons ce cas assez rare.

3.6 Valeur du timer de retransmission

Nous avons choisi 2 secondes comme valeur de RTO car, en négligeant le temps de traitement, cela correspond au délai aller maximal entre un sender et un receiver. Le choix du RTO est une question faisant débat car un plus court RTO permettrait un gain d'efficacité pour toutes les communications avec un délai court et un RTO plus grand pertmettrait d'éviter la rentransmission inutile de paquet lorsque le délai est supérieur à 1 seconde.

A l'idéal nous aurions voulu implémenter un RTO adaptatif sur base du délai mesuré graçe au timestamp mis à jour par le receiver mais pas manque de temps nous avons choisis comme compromis un RTO de 2 secondes.

3.7 Gestion des paquets de type NACK

Lorsque le receiver reçoit un paquet tronqué, il renvoie un paquet de type NACK avec le numéro de séquence du paquet tronqué pour signaler au sender de renvoyer le paquet. Quant au sender, il renvoie en directement le paquet correspondant au NACK reçu afin de gagner en efficacité et ne pas attendre le time out du paquet.

3.8 Cas où le receiver ne peut traiter la connexion

Nous n'avons malheureusement pas su gérér ce cas.

3.9 Partie critique de l'implémentation

Nous avons fait de notre mieux pour optimiser notre implémentation graçe à, par exemple, le retransmission prioritaire des paquets TR ou des paquets pour lequel le sender à reçu au moins 2 ACK à la suite ou encore en évitant d'envoyer trop de paquet inutilement lorsque un paquet timeout. Cependant, ils restent quelques bottleneck tels que la gestion du RTO qui n'est pas optimal. Implémenter un RTO adaptif en fonction du délai moyen et du délai courant avec le receiver serait beaucoup efficace.

4 Tests

4.1 Stratégie

La stratégie utilisée pour les tests a été de les réaliser tôt dans le processus de développement, dans le but de pouvoir imaginer les différents cas limites à l'avance et en tenir compte immédiatement dans notre implémentation. Les tests couvrent le formatage des paquets ainsi que notre structure de données principale, ces librairies sont fondamentales dans le bon fonctionnement de notre implémentation et c'est la raison pour laquelle elles sont massivement couvertes par les tests.

4.2 Tests unitaires

4.2.1 packet_tests.c

Cette suite de tests couvre la partie paquets UDP de l'implémentation, même si cette partie était testable via un exercice INGInious, nous avons ajouté de nombreux tests en local afin de premièrement, faciliter le développement, et deuxièmement, s'assurer que l'implémentation reprend tous les cas limites, ce qui est primordial étant donné l'importance de cette partie dans le bon fonctionnement du projet.

4.2.2 buffer_tests.c

Cette suite de tests couvre quant à elle la partie structure de données de notre implémentation, le buffer du receiver et du sender reposant intégralement sur cette structure, il est primordial qu'elle puisse être fiable. Nous avons ainsi implémenté ces tests conjointement au développement de celle-ci, afin de s'assurer que la structure fonctionnait, pour ainsi éviter les effets de bords sur l'implémentation du sender et du receiver.

4.3 Tests d'intéropérabilité

4.3.1 Groupe 28 : Antoine Demblon - Xavier De Liedekerke

Ce premier test d'intéropérabilité a permis de voir que notre implémentation était relativement robuste, nous avons eu des soucis car leur groupe n'avait pas encore implémenté la stratégie de clôture de connexion, ce qui faisait que les programmes ne terminaient pas leur exécution. Le contenu des tests a également permis de déceller une erreur de conception de leur côté, étant donné que leur numéro de séquence était compris entre $0 <= X <= 248$, ce qui ne respecte pas les spécifications et cela posait donc problème lorsque nous étions le receveur (lorsque nous étions sender cela se déroulait bien grâce au cumulative ack).

4.3.2 Groupe 81 : Antoine X et Justin X

Lors de ce deuxième test nous avons principalement tenté de s'envoyer un fichier d'environ 1 000 000 de paquets, ce qui a permis de trouver une erreur dans notre conception, notre receiver envoyait la taille maximale de notre window et non le nombre de places restantes, ce qui avait pour conséquence d'autoriser le sender à envoyer énormément de paquets à la suite. Cela n'avait pas posé de problème dans les précédents tests car le sender n'envoyait pas autant de paquets simultanément, ici le nombre de paquets était tellement conséquent que cela a révélé des problèmes, ce qui nous est donc bien utile. Nous avons également pu repérer des problèmes de latence important car notre receiver ne renvoyait pas un ACK lorsqu'il recevait des paquets out of window ou corrompu.

4.3.3 Groupe 77 : Brieuc Pierre

Ce troisième et dernier test a été assez bénéfique, nous avons testé nos implémentations dans beaucoup de conditions différentes et les tests se sont tous bien déroulés lorsque nous étions receiver. Lorsque nous étions sender, nous avons trouvé un cas limite qui nous posait problème, si le receiver envoie des acquittements inférieurs à ceux qu'il a déjà envoyé, cela crée des problèmes dans le comportement de notre sender, qui désorganise totalement son buffer d'envoi.

5 Performances

Nous avons effectué des mesures en envoyant un fichier d'une longueur de 317 paquets en comptant le nombre de paquets allaient être renvoyés selon le % de perte.

1. 5% : 346
2. 10% : 380
3. 20% : 456

Il est intéressant de constater que plus le pourcentage de perte augmente, plus le nombre de paquets envoyés augmente, cela s'explique par le fait que le nombre de paquets qui timed out augmente également, c'est donc une piste d'amélioration que nous aurions pu explorer également.

6 Conclusion

Durant les semaines qui ont précédé la remise de ce projet, nous avons pu nous familiariser avec des aspects pratiques des protocoles et de leur implémentation à bas niveau. Ce projet à demandé beaucoup de rigueur et d'organisation dès le départ, c'est pourquoi une bonne structure, des tests robustes et le respect des conventions nous ont grandement aidé. Nous avons également pu tester nos programmes avec d'autres groupes, ce qui a rendu le projet très intéréssant et nous a permis de trouver des failles dans notre implémentation que nous avons pu corriger.